

Q4: Gift Voucher Code Cracking

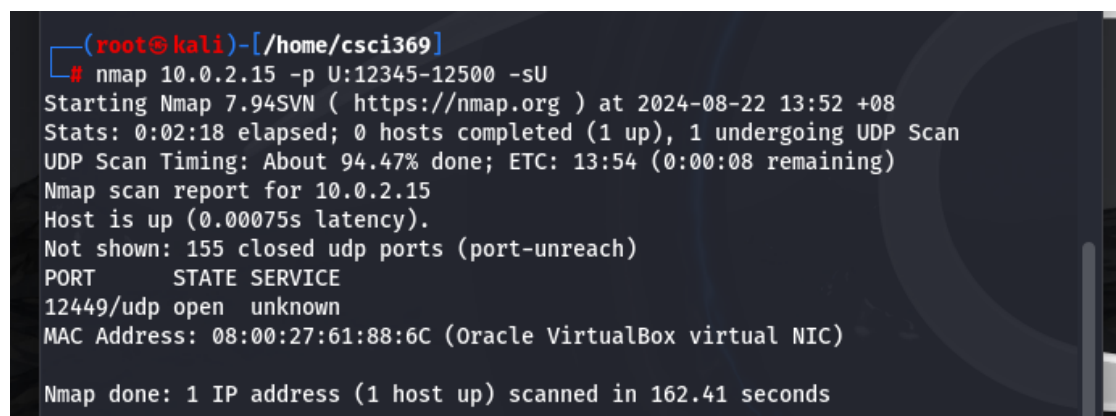
0. Context

A server generates a gift voucher code based on a client ID to the customer. The server does this via a UDP port between 12345 to 12500. It also uses MD5 hash function to generate the code taking (A||ClientID||B) as an input where A is two unknown lowercase letters (aa, ab, ac...) and B is two unknown symbols (##, #!, #\$.).

The server runs on an Ubuntu VM and the server admin failed to close the port for the service.

1. Identifying Open Port

We know that the service uses UDP on a port between 12345 to 12500. As such, we can use nmap to scan for the open port with the option -p to set the range of ports, and -sU for UDP port scan.



```
(root@kali)-[/home/csci369]
# nmap 10.0.2.15 -p U:12345-12500 -sU
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-08-22 13:52 +08
Stats: 0:02:18 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 94.47% done; ETC: 13:54 (0:00:08 remaining)
Nmap scan report for 10.0.2.15
Host is up (0.00075s latency).
Not shown: 155 closed udp ports (port-unreach)
PORT      STATE SERVICE
12449/udp  open  unknown
MAC Address: 08:00:27:61:88:6C (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 162.41 seconds
```

We can identify port **12449** as the open port as seen from the results above.

2. Obtaining Gift Code Voucher

To obtain the gift code voucher, a simple python program is created using the scapy library. With the discovered port, we can generate and send a UDP packet to the specific server IP and port (Ubuntu VM and port 12449). We set the client ID as the load, as well as setting a random source port to receive the response.

```

from scapy.all import *

# Executable Server details
dst_ip = "10.0.2.15"
dst_port = 12449

# Client ID
client_id = "7910939"

# Crafting the UDP packet
source_port = 54321 # random sport
packet = IP(dst=dst_ip)/UDP(sport=source_port, dport=dst_port)/Raw(load=client_id)

# Send a packet and wait for response
response = sr1(packet, timeout=3)

if response:
    print(response.show())
else:
    print("No response received.")

```

By executing the program, we receive the response from the server and obtained our gift voucher code as seen below.

```

(root@kali)-[/home/csci369/Desktop]
# python3 q4.py
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 148
  id        = 10694
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = udp
  chksum    = 0xf87f
  src       = 10.0.2.15
  dst       = 10.0.2.5
  \options  \
###[ UDP ]###
  sport     = 12449
  dport     = 54321
  len       = 128
  chksum    = 0xba26
###[ Raw ]###
  load      = b'Your client ID, 7910939, has been transformed into a secret! Here is y
our voucher code: 6218b6f3f429c2fa1d20e2b04daed3bb'

```

3. Finding the Values of A and B

To generate all possible values of the voucher code, crunch is used. The symbols @@ represents all possible lowercase letters, while ^^ represents all possible symbols. The output is saved to input.txt.

```
(root@kali)-[/home/csci369/Desktop]
# crunch 11 11 -t @7910939^^ -o input.txt
Crunch will now generate the following amount of data: 8833968 bytes
8 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 736164
crunch: 100% completed generating output
```

We save the the gift code voucher into hash.txt obtained in the previous step and start cracking the values using hashcat. The option '-a 0' indicates a dictionary attack, while '-m 0' indicates using an md5 hash.

```
(root@kali)-[/home/csci369/Desktop]
# hashcat -a 0 -m 0 hash.txt input.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian Linux, None+Asserts, RELOC, LLVM 17.0.6, SLEEF, DIST
RO, POCL_DEBUG) - Platform #1 [The pocl project]
=====
* Device #1: cpu-penryn-AMD Ryzen 7 5800HS with Radeon Graphics, 2159/4383 MB (1024 MB alloc
atable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1
```

We get a matching hash with the value 'jj7910939=#'.

```
6218b6f3f429c2fa1d20e2b04daed3bb:jj7910939=#

Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 0 (MD5)
Hash.Target.....: 6218b6f3f429c2fa1d20e2b04daed3bb
Time.Started....: Thu Aug 22 16:14:47 2024 (1 sec)
Time.Estimated...: Thu Aug 22 16:14:48 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (input.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 2174.0 kH/s (0.14ms) @ Accel:512 Loops:1 Thr:1 Vec:4
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 266240/736164 (36.17%)
Rejected.....: 0/266240 (0.00%)
Restore.Point....: 264192/736164 (35.89%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: jj7910939}> -> jk7910939^,
Hardware.Mon.#1...: Util: 27%

Started: Thu Aug 22 16:14:46 2024
Stopped: Thu Aug 22 16:14:50 2024
```

By running a simple python program to check the md5 hash of the output, we can confirm that the values are correct.

```
(csci369@kali)-[~/Desktop]
$ python3 hash_check.py
Voucher Code: 6218b6f3f429c2fa1d20e2b04daed3bb
Calculated Hash: 6218b6f3f429c2fa1d20e2b04daed3bb
```

Thus, we can conclude that the values of A and B are:

A: jj

B: #=

Note: All programs used here can also be found in the folder Q4.